Sum()

SELECT

product\_id,

sale\_date,

sales,

SUM(sales) OVER (PARTITION BY product\_id ORDER BY sale\_date) as running\_total

FROM sales\_table;

Case Scenario

SELECT

patient\_id,

visit\_date,

cost,

SUM(cost) OVER (PARTITION BY patient\_id ORDER BY visit\_date) as cumulative\_cost

FROM medical\_records;

Count()

SELECT customer\_name, order\_date, COUNT(order\_id) OVER (PARTITION BY customer\_name)

FROM orders;

Case Scenario

SELECT User\_ID, COUNT(Post\_ID) OVER(PARTITION BY User\_ID) AS Post\_Count FROM Posts

Avg()

SELECT Date, StockPrice, AVG(StockPrice) OVER (ORDER BY Date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS MovingAvg FROM StockData;

Note : ROWS BETWEEN 6 PRECEDING AND CURRENT ROW syntax defines the window size. It includes the current row and the 6 preceding rows. In other words, it calculates the average over the last 7 days (including the current day).

Case Scenario

SELECT store\_location, AVG(daily\_sales) OVER (PARTITION BY store\_location) AS avg\_sales

FROM sales\_data

WHERE date >= '2023-02-01' AND date <= '2023-02-07'

ORDER BY store\_location;

SELECT employee\_id, employee\_name, department, salary, ROW\_NUMBER() OVER (PARTITION BY department ORDER BY salary DESC) AS rank

FROM employees;

Case Scenario

SELECT

Shipment\_ID,

Carrier,

Origin,

Destination,

Shipment\_Date,

Arrival\_Date,

ROW\_NUMBER() OVER (ORDER BY Arrival\_Date - Shipment\_Date) AS 'Delivery Rank'

FROM shipment\_data

SELECT

salesperson\_name,

COUNT(\*) AS cars\_sold,

ROW\_NUMBER() OVER (ORDER BY COUNT(\*) DESC) AS sales\_rank

FROM sales

GROUP BY salesperson\_name;

Rank()and Dense\_rank()

SELECT

name,

score,

RANK() OVER (ORDER BY score DESC) AS rank,

DENSE\_RANK() OVER (ORDER BY score DESC) AS denserank

FROM

students;

Lead() and Lag() function

SELECT Month, Sales,

Lag(Sales, 1, 0) OVER (ORDER BY Month) AS PrevMonthSales,

Lead(Sales, 1, 0) OVER (ORDER BY Month) AS NextMonthSales

FROM Sales;

Case Scenario

SELECT Date, Orders,

Orders - Lag(Orders, 1, 0) OVER (ORDER BY Date) AS DailyOrderChange

FROM OrdersData;

SELECT Date, Revenue,

(Lead(Revenue, 1, 0) OVER (ORDER BY Date) - Revenue) / Revenue \* 100 AS DailyRevenueChange

FROM OrdersData;

SELECT page\_url,

1 - (COUNT(\*) FILTER (WHERE Lag(page\_url, 1, '') OVER (PARTITION BY session\_id ORDER BY time\_spent) = page\_url) / COUNT(\*) OVER (PARTITION BY session\_id)) AS bounce\_rate

FROM user\_sessions

GROUP BY page\_url

ORDER BY bounce\_rate DESC;

Ntile() function

SELECT sale\_id, amount, NTILE(4) OVER (ORDER BY amount) AS bucket

FROM sales;

Case Scenario

SELECT user\_id, group, pages\_visited, NTILE(4) OVER (ORDER BY pages\_visited) AS bucket

FROM user\_data;